Exhibit B

Appendix A

```
//----------------------------------------------------------------------
//
//  Copyright (C) 1997 Canon Information Systems, All Rights Reserved.
//
//  $Workfile:   uisheet.h  $
//  $Revision:   1.57  $
//  $Author:   hals  $
//  $Date:   Sep 12 1997 10:07:54  $
//
//  Description
//      This is the header file for the CUiSheet class, and all of its
//      individual Property Pages.
//
//  $Log:   Q:/twain4/archive/src/ui/uisheet.h_V  $
//
//    Rev 1.57   Sep 12 1997 10:07:54   hals
// d-942: Clear histogram display during ClearPreview
//
//    Rev 1.56   Sep 04 1997 11:02:12   lgoldsmi
// Add a get and overloaded method for set max values to deal
// with rotation.
//
//    Rev 1.55   Sep 04 1997 08:52:54   lgoldsmi
// Add methods and member variables for the width and height display
// on the main page.  This will allow the width and height to not be
// susceptible to round-off that is needed in iop.
//
//    Rev 1.54   Aug 20 1997 13:24:46   hals
// d-591: Provide message handler to allow DDUI to display status messages
//
//    Rev 1.53   Aug 05 1997 15:42:40   hals
// d-661: Adjust tone control visibility when color mode changed
//
//    Rev 1.52   Jul 28 1997 10:51:34   hals
// d-499: Prevent changing active property page if error occurred.
//
//    Rev 1.51   Jul 18 1997 10:39:28   hals
// Prevent scaling field from beeping on invalid data
//
//    Rev 1.50   Jul 18 1997 10:06:46   hals
// d-459: Fix edit entry fields so that Enter acts like Tab in all cases
//
//    Rev 1.49   Jul 16 1997 10:53:18   hals
// d-420, d-482: Improve error detection and reporting for gamma edit field
//
//    Rev 1.48   Jul 15 1997 15:43:12   hals
// d-469: Put x and y resolution in status bar if they are different
//
//    Rev 1.47   Jul 08 1997 13:09:30   hals
// (d-369) Fix resolution default when resolution exceeds scanner max
//
//    Rev 1.46   Jul 03 1997 13:30:52   hals
// (d-279) Remove tone tab if bilevel mode
//
//    Rev 1.45   Jun 26 1997 14:38:32   hals
// (d-192) Fix highlighting when tabbing between width and height fields
//
//    Rev 1.44   Jun 20 1997 10:03:26   hals
// Added HWnd method to CPageScan
//
```

```
//    Rev 1.43   Jun 16 1997 14:08:08   hals
// Added method to access output scale
//
//    Rev 1.42   Jun 13 1997 13:59:48   hals
// (d-166, 167) Added Alt-P/Alt-S accelerators to property sheet
//
//    Rev 1.41   Jun 11 1997 13:18:12   hals
// (d-187) Put About menu item in system menu, remove rollover
//
//    Rev 1.40   Jun 06 1997 14:12:40   hals
// (d-201) Make remaining edit fields CSEdit so Enter acts like Tab
//
//    Rev 1.39   Jun 06 1997 13:17:08   hals
// (d-209) Force update of preview image when moving from tone mode to another
//
//    Rev 1.38   Jun 02 1997 13:29:00   hals
// (d-122) Update histogram information if new preview performed.
//
//    Rev 1.37   May 22 1997 14:10:28   hals
// (d-60) Do not allow modification or deletion of standard resolutions
//
//    Rev 1.36   May 21 1997 16:35:20   hals
// (AR#91) Delay posting error messages detected in OnKillFocus methods
// (AR#48) Allow ENTER key to terminate edit field entries
//
//    Rev 1.35   May 16 1997 14:01:44   hals
// Do validity checks on output scale field
//
//    Rev 1.34   May 16 1997 13:43:34   hals
// Disable Tone tab if B&W or TextEnhanced color mode selected
//
//    Rev 1.33   May 13 1997 10:42:22   hals
// Display final image size instead of preview image size (#56)
//
//    Rev 1.32   May 09 1997 14:37:50   hals
// Added Canon rollover copyright/version display
//
//    Rev 1.31   08 May 1997 16:38:26   KGrigsby
// Added Public Member function UpdateResolutionValues().
//
//    Rev 1.30   Apr 28 1997 16:15:38   hals
// Support typing into Width and Height fields on Main tab
//
//    Rev 1.29   Apr 25 1997 14:52:22   hals
// Removed dead code
//
//    Rev 1.28   Apr 23 1997 14:07:42   hals
// Added auto-level support
//
//    Rev 1.27   Apr 21 1997 14:40:06   hals
// Added gamma value edit field
//
//    Rev 1.26   Apr 18 1997 13:51:02   hals
// Performance improvements
//
//    Rev 1.25   Apr 16 1997 11:17:38   hals
// Support for separate color channel curves
//
//    Rev 1.24   Apr 03 1997 12:51:40   hals
// Added CGammaWnd member to CPageTone
```

```
//
//    Rev 1.23   Mar 20 1997 16:07:28   hals
// Added ResizeDialogButton, SetUIModeButton
//
//    Rev 1.22   11 Mar 1997 15:29:32   Eas
// try again at changing the tab name
//
//    Rev 1.21   11 Mar 1997 13:09:44   Eas
// fixes for new scanner interface
//
//    Rev 1.19   Mar 03 1997 13:51:14   hals
// Force all bitmaps buttons to a fixed size
//
//    Rev 1.18   Feb 20 1997 14:33:58   hals
// Removed CPageImage class
//
//    Rev 1.17   Feb 19 1997 15:38:18   hals
// Added LoadCurve/SaveCurve/OnCustomCurve methods
//
//    Rev 1.16   Feb 18 1997 12:34:44   hals
// Added Load/Save handlers
//
//    Rev 1.15   Feb 17 1997 13:30:18   hals
// Added support for color match and text enhance options
//
//    Rev 1.14   Feb 13 1997 10:50:06   hals
// Added tooltip support
//
//    Rev 1.13   Feb 05 1997 14:28:26   hals
// Handle SysColorChange
//
//    Rev 1.12   Jan 31 1997 09:57:18   hals
// Added Scanner page
//
//    Rev 1.11   Jan 17 1997 13:45:04   hals
// Remove dead method declaration (PopulateResolutionsCombo)
//
//    Rev 1.10   Jan 15 1997 13:11:38   hals
// Consult scanner capabilities in loading Resolutions combobox
// Removed Destinations combobox
//
//    Rev 1.9   Jan 13 1997 13:53:38   hals
// Added methods to support destination changes
//
//    Rev 1.8   Jan 10 1997 14:08:56   hals
// Added support for rulers.
//
//----------------------------------------------------------------------------

#include "pub_iop.h"
#include "curve.h"
#include "histctrl.h"
#include "scuidisp.h"
#include "edit.h"

class CGammaWnd;

#define BUTTON_SIZE 23

#define PERMANENT_RES  0x00008000L
```

```
#define WM_FIELDERROR   WM_USER+101
#define WM_STATUSUPDATE WM_USER+105

#define US_WIDTHFIELD       1
#define US_HEIGHTFIELD      2

enum ResizeMode { RM_CENTER, RM_NEW, RM_OLD };
void ResizeDialogButton ( CButton& btn, int x = -1, int y = -1, ResizeMode mode = RM_NEW );


/////////////////////////////////////////////////////////////////////////////
// CPageMain dialog

class CPageMain : public CPropertyPage
{
    DECLARE_DYNCREATE(CPageMain)

// Construction
public:
    CPageMain();
    ~CPageMain();

// Dialog Data
    //{{AFX_DATA(CPageMain)
    enum { IDD = IDD_PAGE_MAIN };
    CButton m_lockBtn;
    CStatic m_outputSize;
    CComboBox   m_unitsCB;
    CComboBox   m_cbResolutions;
    CComboBox   m_comboColor;
    CComboBox   m_PresetSizeCombo;
    //}}AFX_DATA

    CSNumEdit   m_scale;
    CSNumEdit   m_editHeight;
    CSNumEdit   m_editWidth;

// Operations
public:
    void LoadColorCombo();
    void LoadResolutionCombo ( CString* select = NULL );
    void InitResolutionRegistry();
    void UpdateResolutionValues();
    void UpdateSizeDisplay ( UINT whichField = US_WIDTHFIELD | US_HEIGHTFIELD );
    void UpdateSelRect ( CRect& imageRect );
    void UpdateStatusBar();
    void GetFinalResolution ( DWORD* dpiX, DWORD* dpiY ) { *dpiX = m_dpiX, *dpiY = m_dpiY;
};
    void SetFinalResolution();
    void LoadBitmaps();
    void StringToSize ( CString& strWidth, CString& strHeight, int& pixelWidth, int& pixelH
eight );
    void PostPendingError ( UINT strID, CString* str, CWnd& wnd );
    double GetScale() { return m_dScale; };
    void UpdateResolutionDisplay();

    // Methods to maintain width and height display integrity without
    // getting changes due to round-off problems.
    void SetDimensionsForDisplay(LONG dWidth, LONG dHeight);
```

```
    void SetToPhysicalSizeDisplay();
    void GetDimensionsForDisplay(LONG *pWidth, LONG *pHeight);
    void GetMaxDimensionValues(LONG *pWidth, LONG *pHeight);
    void SetMaxDimensionValues();
    void SetMaxDimensionValues(LONG dWidth, LONG dHeight);
    void SetToMaxSizeDisplay();

// Overrides
    afx_msg void OnSysColorChange();

    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPageMain)
    public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    virtual BOOL OnKillActive();
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    BOOL        m_bResolutionTypein;
    CRect       m_selRect;
    double      m_dScale;
    DWORD       m_dpiX;
    DWORD       m_dpiY;
    BOOL        m_bLock;
    BOOL        m_bErrorPending;

    SIZE        m_DisplaySize;          // Current cx and cy values being displayed.
    SIZE        m_MaxDisplaySize;       // Upper limit for cx and cy display values.

    CToolTipCtrl*   m_pToolTip;

protected:
    // Generated message map functions
    //{{AFX_MSG(CPageMain)
    virtual BOOL OnInitDialog();
    afx_msg void OnSelchangeColor();
    afx_msg void OnClickedLock();
    afx_msg void OnKillfocusResolution();
    afx_msg void OnSelchangePgmainUnitscombo();
    afx_msg void OnSelchangePgmainResolutioncombo();
    afx_msg void OnEditchangePgmainResolutioncombo();
    afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnKillfocusPgmainScalingedit();
    afx_msg void OnKillfocusHeightEdit();
    afx_msg void OnKillfocusWidthEdit();
    //}}AFX_MSG
    afx_msg BOOL OnToolTipNotify(UINT id, NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg LRESULT OnFieldError ( WPARAM wParam, LPARAM lParam );
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////////
// CPageTone dialog

#define NUM_CHANNELS    4
#define CH_MASTER       0
```

```
#define CH_RED          1
#define CH_GREEN        2
#define CH_BLUE         3

void        LoadButtonBitmap ( CWnd* pWnd, UINT nID, UINT nBmp );
void        LoadStaticBitmap ( CWnd* pWnd, UINT nID, UINT nBmp );

class CPageTone : public CPropertyPage
{
    DECLARE_DYNCREATE(CPageTone)

// Construction
public:
    CPageTone();
    ~CPageTone();

// Dialog Data
    //{{AFX_DATA(CPageTone)
    enum { IDD = IDD_PAGE_TONE };
    CButton m_autoBtn;
    CComboBox   m_channelCB;
    CButton m_brightBtn;
    CButton m_gammaBtn;
    CButton m_histBtn;
    CButton m_customBtn;
    CButton m_loadBtn;
    CButton m_resetBtn;
    CButton m_saveBtn;
    CButton m_shadowBtn;
    CButton m_midtoneBtn;
    CButton m_hiliteBtn;
    CComboBox   m_presetCB;
    CSliderCtrl m_sliderGamma;
    CSliderCtrl m_sliderContrast;
    CSliderCtrl m_sliderBrightness;
    //}}AFX_DATA

    CSNumEdit   m_shadowEdit;
    CSNumEdit   m_midtoneEdit;
    CSNumEdit   m_hiliteEdit;
    CSNumEdit   m_gammaEdit;

// Overrides
public:
    void        SetShadow ( UCHAR value, BOOL update = FALSE );
    void        SetMidtone ( UCHAR value, BOOL update = FALSE );
    void        SetHighlight ( UCHAR value, BOOL update = FALSE );
    void        ComputeHistoCurve ( int channel );
    int         CurrentChannel() { return m_nChannel; };
    void        SetUIModeButton ( CScuiDispatch::UIMODE uiMode, BOOL value );
    void        UpdateControls();
    void        RecomputeCurve ( int channel );
    void        DownloadCurve ( int channel );
    void        DownloadAllCurves();
    void        CalculateWhiteBlackPoints ( CHistogram& histogram, int& whitePt, int& black
Pt );
    void        AutoAdjust ( BOOL bDownload = FALSE );
    BOOL        AutoLevel() { return m_bAuto; };
    void        SetAutoLevel ( BOOL bAuto );
    void        UpdateHistogram();
```

```
    void        PostPendingError ( UINT strID, CString* str, CWnd& wnd );
    void        AdjustToneControls();
    void        InvalidateCurveWindow();

    afx_msg void OnSysColorChange();

    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPageTone)
    public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    virtual BOOL OnKillActive();
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    int         m_toneMode;
    BOOL        m_bAuto;
    int         m_nChannel;

    BOOL        m_bErrorPending;

    DWORD       m_nBright[NUM_CHANNELS];
    DWORD       m_nContrast[NUM_CHANNELS];
    DWORD       m_nGamma[NUM_CHANNELS];

    CurveArray  m_aCurve[NUM_CHANNELS];
    CHistogram  m_histogram[NUM_CHANNELS];
    DWORD       m_maxHistValue;

    BYTE        m_shadow[NUM_CHANNELS];
    BYTE        m_midtone[NUM_CHANNELS];
    BYTE        m_highlight[NUM_CHANNELS];

    CToolTipCtrl*   m_pToolTip;
    CCurveEditWnd*  m_pCurveWnd;
    CGammaWnd*      m_pGammaWnd;
    CHistogramControls*     m_pHistCtrls;

    void        LoadButtonBitmaps();
    void        LoadStaticBitmaps();
    void        PaintBitmap ( CDC* pdc, UINT nID, HBITMAP hBmp );

    afx_msg BOOL OnToolTipNotify(UINT id, NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg LRESULT OnCustomCurve ( WPARAM wParam, LPARAM lParam );

protected:
    // Generated message map functions
    //{{AFX_MSG(CPageTone)
    virtual BOOL OnInitDialog();
    afx_msg BOOL OnSelectToneMode( UINT nID );
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnPgtoneResetbutton();
    afx_msg void OnSelchangePgtonePresets();
    afx_msg void OnPgtoneLoadbutton();
    afx_msg void OnPgtoneSavebutton();
    afx_msg void OnPgtoneHighlightbutton();
    afx_msg void OnPgtoneMidtonebutton();
```

```
    afx_msg void OnPgtoneShadowbutton();
    afx_msg void OnKillfocusPgtoneHighlightedit();
    afx_msg void OnKillfocusPgtoneMidtoneedit();
    afx_msg void OnKillfocusPgtoneShadowedit();
    afx_msg void OnSelchangeChannelcb();
    afx_msg void OnKillfocusPgtoneGammaedit();
    //}}AFX_MSG
    afx_msg LRESULT OnFieldError ( WPARAM wParam, LPARAM lParam );
    DECLARE_MESSAGE_MAP()

    friend CGammaWnd;

};

HBITMAP LoadSysColorBitmap ( UINT nID );

/////////////////////////////////////////////////////////////////////////////
// CPagePref dialog

class CPagePref : public CPropertyPage
{
    DECLARE_DYNCREATE(CPagePref)

// Construction
public:
    CPagePref();
    ~CPagePref();

// Dialog Data
    //{{AFX_DATA(CPagePref)
    enum { IDD = IDD_PAGE_PREF };
    CButton m_textEnhance;
    CButton m_colorMatch;
    //}}AFX_DATA


// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPagePref)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPagePref)
    afx_msg void OnColorMatch();
    afx_msg void OnTextEnhance();
    virtual BOOL OnInitDialog();
    afx_msg void OnChangeProfile();
    //}}AFX_MSG
    afx_msg LRESULT OnSysColorChange ( WPARAM wParam, LPARAM lParam );
    DECLARE_MESSAGE_MAP()

};

/////////////////////////////////////////////////////////////////////////////
// CPageScan dialog
```

```
class CPageScan : public CPropertyPage
{
    DECLARE_DYNCREATE(CPageScan)

// Construction
public:
    CPageScan();
    ~CPageScan();
    void SetPageNum(int i);
    HWND    HWnd() { return m_devHWND; };

// Dialog Data
    //{{AFX_DATA(CPageScan)
    enum { IDD = IDD_PAGE_SCAN };
        // NOTE - ClassWizard will add data members here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_DATA


// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPageScan)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    int     m_devPageNum;

    HWND    m_devHWND;

    // Generated message map functions
    //{{AFX_MSG(CPageScan)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    afx_msg LRESULT OnStatusUpdate ( WPARAM wParam, LPARAM lParam );
    DECLARE_MESSAGE_MAP()

};
/////////////////////////////////////////////////////////////////////////////
// CuiSheet

class CuiSheet : public CPropertySheet
{
    DECLARE_DYNAMIC(CuiSheet)

// Construction
public:
    CuiSheet();

// Attributes
public:
    // pages
    CPageMain   m_pageMain;
    CPageTone   m_pageTone;
    //CPageImage  m_pageImage;
    CPagePref   m_pagePref;
    //CPageScan m_pageScan;
```

```
    CToolTipCtrl*   m_pTipCtrl;
    HACCEL      m_hAccel;

// Operations
public:
    BOOL SetPreviewBitmap( LPBITMAPINFOHEADER pBmp );
    void OnSysColorChange();
    BOOL HasToneTab() { return m_bHasToneTab; };
    void RemoveToneTab();
    void RestoreToneTab();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CuiSheet)
    public:
    virtual BOOL Create( CWnd* pParentWnd );
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CuiSheet();

protected:
    CSize       m_sizeSheet;
    BOOL        m_bHasToneTab;

    // Generated message map functions
protected:
    //{{AFX_MSG(CuiSheet)
    afx_msg LRESULT OnSizeParent(WPARAM wParam, LPARAM lParam);
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    afx_msg void OnPrescan();
    afx_msg void OnScan();
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

// global pointer to one and only sheet object
extern CuiSheet* g_pSheet;
```

```
//----------------------------------------------------------------------------
//
//  Copyright (C) 1997 Canon Information Systems, All Rights Reserved.
//
//  $Workfile:   PAGETONE.CPP  $
//  $Revision:   1.61  $
//  $Author:   hals  $
//  $Date:   Sep 16 1997 14:21:12  $
//
//  Description
//      This is the implementation for the CPageTone class, which provides
//      the user access to the curve altering capabilities.
//
//  $Log:   Q:/twain4/archive/src/ui/pagetone.cpp  $
//
//    Rev 1.61   Sep 16 1997 14:21:12   hals
// d-958: Make tone edit fields produce full errors to prevent button problems
//
//    Rev 1.60   Sep 12 1997 13:36:34   hals
// De-couple tone tab from BP1 brightness / contrast controls
// Re-arrange how device-specific tabs are re-initialized when tone tab removed
//
//    Rev 1.59   Sep 05 1997 13:09:40   hals
// d-388: Default to device-specific directory for Load/Save curve operations
//
//    Rev 1.58   Aug 28 1997 08:41:06   hals
// d-839: Restrict tone values to 255
//
//    Rev 1.57   Aug 27 1997 10:54:52   hals
// d-485: Restore previous channel settings after using AutoTone
//
//    Rev 1.56   Aug 14 1997 10:22:30   hals
// d-705: Ignore auto-tone requests if image is not color
//
//    Rev 1.55   Aug 05 1997 15:42:58   hals
// d-661: Adjust tone control visibility when color mode changed
//
//    Rev 1.54   Aug 05 1997 14:10:24   hals
// d-205, d-215, d-638 Fix various histogram control value problems
//
//    Rev 1.53   Jul 28 1997 10:51:54   hals
// d-499: Prevent changing active property page if error occurred.
//
//    Rev 1.52   Jul 25 1997 10:45:48   hals
// d-537: Allow localization of decimal numeric formats
//
//    Rev 1.51   Jul 22 1997 13:35:10   hals
// d-545: Check for presence of bitmap before attempting to update histogram
//
//    Rev 1.50   Jul 21 1997 13:53:16   hals
// d-523: Check for presence of bitmap before auto-toning
//
//    Rev 1.49   Jul 18 1997 10:07:18   hals
// d-459: Fix edit entry fields so that Enter acts like Tab in all cases
//
//    Rev 1.48   Jul 17 1997 15:24:54   hals
// d-500: Force gamma value into dd.d format in edit field
//
//    Rev 1.47   Jul 17 1997 15:11:10   hals
// d-484: Check for bitmap before attempting to compute histogram
```

```
//
//    Rev 1.46   Jul 16 1997 14:50:28   hals
// d-489: Hide Reset button if AutoTone selected
//
//    Rev 1.45   Jul 16 1997 10:53:18   hals
// d-420, d-482: Improve error detection and reporting for gamma edit field
//
//    Rev 1.44   Jul 11 1997 13:34:50   hals
// (d-436) Limit text to 3 digits in black/gray/white point edit fields
//
//    Rev 1.43   Jul 09 1997 12:35:22   hals
// (d-419) Fix update of image and graph when selecting AutoTone
//
//    Rev 1.42   Jul 03 1997 13:31:12   hals
// (d-321, 385) Disable color channel combobox if grayscale
//
//    Rev 1.41   Jun 13 1997 14:01:50   hals
// (d-196) Improve coloring of histogram curves
//
//    Rev 1.40   Jun 06 1997 14:13:10   hals
// (d-201) Make remaining edit fields CSEdit so Enter acts like Tab
//
//    Rev 1.39   Jun 06 1997 13:17:36   hals
// (d-209) Force update of preview image when moving from tone mode to another
//
//    Rev 1.38   Jun 05 1997 13:59:54   hals
// (d-189) Force focus to first edit control on certain tone pages
//
//    Rev 1.37   Jun 03 1997 09:35:46   hals
// (d-138, d-139) Add labels to unlabeled tone page controls
//
//    Rev 1.36   Jun 02 1997 13:29:18   hals
// (d-122) Update histogram information if new preview performed.
//
//    Rev 1.35   May 30 1997 13:17:52   hals
// (d-159) Check range of white/mid/black point edit fields
//
//    Rev 1.34   May 16 1997 13:44:00   hals
// Disable Tone tab if B&W or TextEnhanced color mode selected
//
//    Rev 1.33   Apr 30 1997 15:15:36   hals
// Retain midtone proportionality when changing white/black point
//
//    Rev 1.32   Apr 29 1997 15:47:34   hals
// Turn off RGB channel select if in Auto mode
//
//    Rev 1.31   Apr 28 1997 16:15:56   hals
// Add tooltip for RGB channel dropdown
//
//    Rev 1.30   Apr 25 1997 14:52:56   hals
// Fixed bitmap painting
// Completed implementation of AutoLevel
//
//    Rev 1.29   Apr 23 1997 14:43:42   hals
// Turn off refresh of preview bitmap until last curve is downloaded
//
//    Rev 1.28   Apr 23 1997 14:07:46   hals
// Added auto-level support
//
//    Rev 1.27   Apr 21 1997 14:40:34   hals
```

```
// Added gamma value edit field
//
//    Rev 1.26   Apr 18 1997 13:51:08   hals
// Performance improvements
//
//    Rev 1.25   Apr 16 1997 11:17:40   hals
// Support for separate color channel curves
//
//    Rev 1.24   Apr 03 1997 12:52:04   hals
// Display gamma curves in separate window
//
//    Rev 1.23   Mar 20 1997 16:08:12   hals
// Added SetUIModeButton, improved ResizeDialogButton
//
//----------------------------------------------------------------------

#include "stdafx.h"
#include "mainfrm.h"
#include "Scui.h"
#include "uiSheet.h"
#include "ScuiDisp.h"
#include "gammawnd.h"
#include "imgenh.h"
#include "picwnd.h"
#include "scuiview.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


#define     HISTOGRAM_SIZE      256

C_ImageEnhanceInterface     ImageIF;

BOOL OnToolTipNotify ( NMHDR* pNMHDR );

// this array translates the current channel to the appropriate CCAP option
int     ChannelToCap[NUM_CHANNELS] = { CCAP_CURVE, CCAP_CURVE_RED, CCAP_CURVE_GREEN, CCAP_C
URVE_BLUE };


// undocumented but public routine in MFC for loading a bitmap, and transforming the colors
// so that they match the user's currently selected system colors
HBITMAP AFXAPI AfxLoadSysColorBitmap ( HINSTANCE hInst, HRSRC hRsrc, BOOL bMono );


//----------------------------------------------------------------------
//
//  Function:   LoadSysColorBitmap
//
//  Purpose :   Loads a bitmap, and calls the Afx function that will map all
//              of the grays of the original bitmap into the currently-
//              selected system colors
//
//  Returns :   Handle to new bitmap
//
//----------------------------------------------------------------------
```

```
HBITMAP LoadSysColorBitmap ( UINT nBmp )
{
    // must load bitmap, and map its colors into the current system colors
    // fortunately, there is a function in the CToolbar source code for doing this
    HRSRC   hRsrc = FindResource ( AfxGetResourceHandle(), MAKEINTRESOURCE(nBmp), RT_BITMAP
);
    return AfxLoadSysColorBitmap ( AfxGetInstanceHandle(), hRsrc, FALSE );

}   // LoadSysColorBitmap

//-----------------------------------------------------------------------
//
//  Function:   LoadStaticBitmap
//
//  Purpose :   Loads a static control bitmap, remaps the colors to the current
//              system colors, then sends the new bitmap to the static control
//
//-----------------------------------------------------------------------

void LoadStaticBitmap ( CWnd* pWnd, UINT nID, UINT nBmp )
{
    HBITMAP hBitmap = LoadSysColorBitmap ( nBmp );
    pWnd->SendDlgItemMessage ( nID, STM_SETIMAGE, IMAGE_BITMAP, (LPARAM)hBitmap );

}   // LoadStaticBitmap


//-----------------------------------------------------------------------
//
//  Function:   LoadButtonBitmap
//
//  Purpose :   Loads a single button bitmap, remaps the colors to the current
//              system colors, then sends the new bitmap to the button
//
//-----------------------------------------------------------------------

void LoadButtonBitmap ( CWnd* pWnd, UINT nID, UINT nBmp )
{
    HBITMAP hBitmap = LoadSysColorBitmap ( nBmp );
    pWnd->SendDlgItemMessage ( nID, BM_SETIMAGE, IMAGE_BITMAP, (LPARAM)hBitmap );

}   // LoadButtonBitmap


////////////////////////////////////////////////////////////////////////////
// CPageTone property page

IMPLEMENT_DYNCREATE(CPageTone, CPropertyPage)

//-----------------------------------------------------------------------
//
//  Method  :   CPageTone :: CPageTone
//
//  Purpose :   Default constructor
//
//-----------------------------------------------------------------------

CPageTone :: CPageTone() : CPropertyPage(CPageTone :: IDD)
```

```
{
    //{{AFX_DATA_INIT(CPageTone)
    //}}AFX_DATA_INIT
    m_pToolTip = NULL;
    m_pCurveWnd = NULL;
    m_pGammaWnd = NULL;
    m_pHistCtrls = NULL;

    m_maxHistValue = 0;
    m_bAuto = FALSE;
    m_bErrorPending = FALSE;

    m_nChannel = CH_MASTER;

    for ( int chnl = 0; chnl < NUM_CHANNELS; chnl++ ) {
        m_shadow[chnl] = 0;
        m_midtone[chnl] = 128;
        m_highlight[chnl] = 255;
    }

    g_Dispatch->GetCurrentCapabilitySetting(ICAP_BRIGHTNESS, &m_nBright[CH_MASTER]);
    g_Dispatch->GetCurrentCapabilitySetting(ICAP_CONTRAST, &m_nContrast[CH_MASTER]);

}   // CPageTone :: CPageTone


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: ~CPageTone
//
//  Purpose :   Destructor
//
//---------------------------------------------------------------------------

CPageTone :: ~CPageTone()
{
    delete m_pToolTip;

    delete m_pCurveWnd;
    delete m_pGammaWnd;
    delete m_pHistCtrls;

}   // CPageTone :: ~CPageTone


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: DoDataExchange
//
//  Purpose :   Connect page controls to member variables
//
//---------------------------------------------------------------------------

void CPageTone :: DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPageTone)
    DDX_Control(pDX, IDC_PGTONE_AUTORADIO, m_autoBtn);
    DDX_Control(pDX, IDC_CHANNELCB, m_channelCB);
    DDX_Control(pDX, IDC_PGTONE_BRICONTRADIO, m_brightBtn);
```

```
    DDX_Control(pDX, IDC_PGTONE_GAMMARADIO, m_gammaBtn);
    DDX_Control(pDX, IDC_PGTONE_HISTORADIO, m_histBtn);
    DDX_Control(pDX, IDC_PGTONE_CURVERADIO, m_customBtn);
    DDX_Control(pDX, IDC_PGTONE_LOADBUTTON, m_loadBtn);
    DDX_Control(pDX, IDC_PGTONE_RESETBUTTON, m_resetBtn);
    DDX_Control(pDX, IDC_PGTONE_SAVEBUTTON, m_saveBtn);
    DDX_Control(pDX, IDC_PGTONE_SHADOWBUTTON, m_shadowBtn);
    DDX_Control(pDX, IDC_PGTONE_MIDTONEBUTTON, m_midtoneBtn);
    DDX_Control(pDX, IDC_PGTONE_HIGHLIGHTBUTTON, m_hiliteBtn);
    DDX_Control(pDX, IDC_PGTONE_PRESETS, m_presetCB);
    DDX_Control(pDX, IDC_PGTONE_GAMMASLIDER, m_sliderGamma);
    DDX_Control(pDX, IDC_PGTONE_CONTRASTSLIDER, m_sliderContrast);
    DDX_Control(pDX, IDC_PGTONE_BRIGHTNESSSLIDER, m_sliderBrightness);
    //}}AFX_DATA_MAP
}


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: LoadButtonBitmaps
//
//  Purpose  :   Loads all of the button bitmaps associated with this page
//
//---------------------------------------------------------------------------

void CPageTone :: LoadButtonBitmaps()
{
    LoadButtonBitmap ( this, IDC_PGTONE_AUTORADIO, IDB_BTNAUTO );
    LoadButtonBitmap ( this, IDC_PGTONE_BRICONTRADIO, IDB_BTNBRICONT );
    LoadButtonBitmap ( this, IDC_PGTONE_GAMMARADIO, IDB_BTNGAMMA );
    LoadButtonBitmap ( this, IDC_PGTONE_HISTORADIO, IDB_BTNHISTO );
    LoadButtonBitmap ( this, IDC_PGTONE_CURVERADIO, IDB_BTNCURVE );

    LoadButtonBitmap ( this, IDC_PGTONE_SHADOWBUTTON, IDB_PICKBLK );
    LoadButtonBitmap ( this, IDC_PGTONE_MIDTONEBUTTON, IDB_PICKGRAY );
    LoadButtonBitmap ( this, IDC_PGTONE_HIGHLIGHTBUTTON, IDB_PICKWHT );

}   // CPageTone :: LoadButtonBitmaps


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: LoadStaticBitmaps
//
//  Purpose  :   Loads all of the static bitmaps (not associated with buttons)
//               for this page, using the LoadSysColorBitmap function to
//               remap all of the grays to the current system colors
//
//---------------------------------------------------------------------------

void CPageTone :: LoadStaticBitmaps()
{
    LoadStaticBitmap ( this, IDC_PGTONE_CONTRASTPICLEFT, IDB_SLIDECONTLEFT );
    LoadStaticBitmap ( this, IDC_PGTONE_CONTRASTPICRIGHT, IDB_SLIDECONTRIGHT );
    LoadStaticBitmap ( this, IDC_PGTONE_BRIGHTNESSPICLEFT, IDB_SLIDEBRITLEFT );
    LoadStaticBitmap ( this, IDC_PGTONE_BRIGHTNESSPICRIGHT, IDB_SLIDEBRITRIGHT );
    LoadStaticBitmap ( this, IDC_PGTONE_GAMMAPICLEFT, IDB_SLIDEGAMMLEFT );
    LoadStaticBitmap ( this, IDC_PGTONE_GAMMAPICRIGHT, IDB_SLIDEGAMMRIGHT );
    LoadStaticBitmap ( this, IDC_WHITE1, IDB_WHITECIRCLE );
    LoadStaticBitmap ( this, IDC_GRAY1, IDB_GRAYCIRCLE );
```

```
    LoadStaticBitmap ( this, IDC_BLACK, IDB_BLACKCIRCLE );
    LoadStaticBitmap ( this, IDC_GRAY2, IDB_GRAYCIRCLE );
    LoadStaticBitmap ( this, IDC_WHITE2, IDB_WHITECIRCLE );

}   // CPageTone :: LoadStaticBitmaps


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: InvalidateCurveWindow
//
//  Purpose :   Invalidates the interior of the curve window, forcing repaint
//
//---------------------------------------------------------------------------

void CPageTone :: InvalidateCurveWindow()
{
    if ( m_pGammaWnd != NULL && ::IsWindow ( m_pGammaWnd->m_hWnd ) ) {
        m_pGammaWnd->Invalidate();
    }

}   // CPageTone :: InvalidateCurveWindow


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: SetShadow
//
//  Purpose :   Update shadow value, redrawing curve if requested
//
//---------------------------------------------------------------------------

void CPageTone :: SetShadow ( UCHAR value, BOOL update )
{
    m_shadow[m_nChannel] = value;
    SetDlgItemInt ( IDC_PGTONE_SHADOWEDIT, value );
    if ( update ) {
        m_pHistCtrls->SetShadow ( value );
        ComputeHistoCurve ( m_nChannel );
    }

}   // CPageTone :: SetShadow


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: SetMidtone
//
//  Purpose :   Update midtone value, redrawing curve if requested
//
//---------------------------------------------------------------------------

void CPageTone :: SetMidtone ( UCHAR value, BOOL update )
{
    m_midtone[m_nChannel] = value;
    SetDlgItemInt ( IDC_PGTONE_MIDTONEEDIT, value );
    if ( update ) {
        m_pHistCtrls->SetMidtone ( value, TRUE );
        ComputeHistoCurve ( m_nChannel );
    }
```

```
}   // CPageTone :: SetMidtone


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: SetHighlight
//
//  Purpose  :   Update highlight value, redrawing curve if requested
//
//---------------------------------------------------------------------------

void CPageTone :: SetHighlight ( UCHAR value, BOOL update )
{
    m_highlight[m_nChannel] = value;
    SetDlgItemInt ( IDC_PGTONE_HIGHLIGHTEDIT, value );
    if ( update ) {
        m_pHistCtrls->SetHighlight ( value );
        ComputeHistoCurve ( m_nChannel );
    }

}   // CPageTone :: SetHighlight


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: ComputeHistoCurve
//
//  Purpose  :   Compute the spline curve for the histogram page, using the
//               current values specified for black, gray and white points
//
//---------------------------------------------------------------------------

void CPageTone :: ComputeHistoCurve ( int channel )
{
    CSpline     spline;
    CPoint      histPts[3];

    // add histogram control points
    histPts[0] = CPoint ( m_shadow[channel], 0 );
    histPts[1] = CPoint ( m_midtone[channel], 128 );
    histPts[2] = CPoint ( m_highlight[channel], 255 );

    // calculate curve
    spline.CalculateGraph ( m_aCurve[channel], histPts, 3 );

    // download new curve to IOP
    DownloadCurve ( channel );

    // invalidate curve window, forcing repaint of new data
    InvalidateCurveWindow();

}   // CPageTone :: ComputeHistoCurve


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: SetUIModeButton
//
//  Purpose  :   Sets the clicked state of the current UI tool button to the
```

```
//              desired value
//
//---------------------------------------------------------------------------

void CPageTone :: SetUIModeButton ( CScuiDispatch::UIMODE uiMode, BOOL value )
{
    switch ( uiMode ) {
        case CScuiDispatch::uimodeShadow:
            m_shadowBtn.SetCheck ( value );
            break;
        case CScuiDispatch::uimodeMidtone:
            m_midtoneBtn.SetCheck ( value );
            break;
        case CScuiDispatch::uimodeHilite:
            m_hiliteBtn.SetCheck ( value );
            break;
    }

}   // CPageTone :: SetUIModeButton


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: UpdateControls
//
//  Purpose :   Update slider and other control settings to reflect the
//              currently active channel
//
//---------------------------------------------------------------------------

void CPageTone :: UpdateControls()
{
    switch ( m_toneMode ) {

    case IDC_PGTONE_BRICONTRADIO:
        m_sliderBrightness.SetPos ( (int)m_nBright[m_nChannel] );
        m_sliderContrast.SetPos ( (int)m_nContrast[m_nChannel] );
        m_pGammaWnd->Invalidate();
        break;

    case IDC_PGTONE_GAMMARADIO: {
        m_sliderGamma.SetPos ( (int)m_nGamma[m_nChannel] );
        double      dGamma;
        if ( m_nGamma[m_nChannel] <= 50 )
            dGamma = 0.018 * m_nGamma[m_nChannel] + 0.1;    // convert (0-50) to (0.1-1)
        else
            dGamma = 0.18 * m_nGamma[m_nChannel] - 8;       // convert (50-100) to (1-10)
        CString str;
        str.Format ( "%3.1f", dGamma );
        SetDlgItemText ( IDC_PGTONE_GAMMAEDIT, str );
        m_pGammaWnd->Invalidate();
    }   break;

    case IDC_PGTONE_HISTORADIO:
        SetShadow ( m_shadow[m_nChannel] );
        m_pHistCtrls->SetShadow ( m_shadow[m_nChannel], FALSE );
        SetHighlight ( m_highlight[m_nChannel] );
        m_pHistCtrls->SetHighlight ( m_highlight[m_nChannel], FALSE );
        SetMidtone ( m_midtone[m_nChannel] );
        m_pHistCtrls->SetMidtone ( m_midtone[m_nChannel], TRUE );
```

```
        m_pGammaWnd->Invalidate();
        break;

    case IDC_PGTONE_CURVERADIO:
        m_pCurveWnd->SetActiveChannel ( m_nChannel );
        m_pCurveWnd->Invalidate();
        break;
    }

}   // CPageTone :: UpdateControls


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: RecomputeCurve
//
//  Purpose :   Recomputes the tone curve for the specified channel, based on
//              the current tone mode, and insures it is downloaded to IOP
//
//---------------------------------------------------------------------------

void CPageTone :: RecomputeCurve ( int channel )
{
    switch ( m_toneMode ) {

        case IDC_PGTONE_AUTORADIO:
            DownloadCurve ( channel );
            break;

        case IDC_PGTONE_BRICONTRADIO:
            ImageIF.CurveFromBC ( m_aCurve[channel], m_nBright[channel]*2-100, m_nContrast[
channel]*2-100 );
            DownloadCurve ( channel );
            break;

        case IDC_PGTONE_GAMMARADIO: {
            double      dGamma;
            if ( m_nGamma[channel] <= 50 )
                dGamma = 0.018 * m_nGamma[channel] + 0.1;   // convert (0-50) to (0.1-1)
            else
                dGamma = 0.18 * m_nGamma[channel] - 8;      // convert (50-100) to (1-10)
            ImageIF.CurveFromGamma ( m_aCurve[channel], dGamma );
            DownloadCurve ( channel );
        }   break;

        case IDC_PGTONE_HISTORADIO:
            ComputeHistoCurve ( channel );
            break;

        case IDC_PGTONE_CURVERADIO:
            g_Dispatch->SetCurrentCapability ( ChannelToCap[channel], m_pCurveWnd->GetGraph
Data(channel) );
            break;
    }

}   // CPageTone :: RecomputeCurve


//---------------------------------------------------------------------------
//
```

```
//  Method   :   CPageTone :: DownloadCurve
//
//  Purpose  :   Download specified curve to IOP. Called when the curve has
//               has changed, or the tone mode has changed.
//
//----------------------------------------------------------------------------

void CPageTone :: DownloadCurve ( int channel )
{
    CWaitCursor     wait;

    g_Dispatch->SetCurrentCapability ( ChannelToCap[channel], m_aCurve[channel] );

}   // CPageTone :: DownloadCurve


//----------------------------------------------------------------------------
//
//  Method   :   CPageTone :: DownloadAllCurves
//
//  Purpose  :   Download all curves to IOP. Called when the tone mode has
//               been changed, and we are dealing with an entirely new set
//               of curve information
//
//----------------------------------------------------------------------------

void CPageTone :: DownloadAllCurves()
{
    int     minChannel, maxChannel;
    DWORD   dwColor;

    // if in RGB mode, download all channels, otherwise we only need to
    // do the master channel
    g_Dispatch->GetCurrentCapabilitySetting ( CCAP_IMAGE_CLASS, &dwColor );

    if ( dwColor == IOP_IMAGECLASS_RGB ) {

        // if current channel is CH_MASTER, we must load it last
        if ( m_nChannel == CH_MASTER ) {
            for ( int channel = CH_BLUE; channel >= CH_MASTER; channel-- ) {
                g_Dispatch->EnableRefresh ( channel == CH_MASTER );
                RecomputeCurve ( channel );
            }
        } else {
            for ( int channel = CH_MASTER; channel <= CH_BLUE; channel++ ) {
                g_Dispatch->EnableRefresh ( channel == CH_BLUE );
                RecomputeCurve ( channel );
            }
        }

    } else {
        // just download the master (gray) channel
        g_Dispatch->EnableRefresh ( TRUE );
        RecomputeCurve ( CH_MASTER );
    }

    minChannel = CH_MASTER;
    maxChannel = dwColor == IOP_IMAGECLASS_RGB ? CH_BLUE : CH_MASTER;

    // since this was called due to a tone mode change, the local curve arrays
```

```
    // do not contain the necessary values for the new tone mode - recalculate
    for ( int channel = minChannel; channel <= maxChannel; channel++ ) {

        // turn off refreshing of the preview bitmap until the last curve is being sent
        g_Dispatch->EnableRefresh ( channel == maxChannel );

    }

}   // CPageTone :: DownloadAllCurves


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: CalculateWhiteBlackPoints
//
//  Purpose :   Given a histogram, calculate the white and black points that
//              should be used in order to perform auto-leveling
//
//---------------------------------------------------------------------------

void CPageTone :: CalculateWhiteBlackPoints ( CHistogram& histogram, int& whitePt, int& bla
ckPt )
{
    int     j, k;
    DWORD   total, frac, cnt;
    DWORD*  pEntry;

    // find total number of hits for this channel
    pEntry = &histogram[0];
    total = 0;
    for ( j = 0; j < HISTOGRAM_SIZE; j++ )
        total += *pEntry++;

    // find black 0.5% threshold point
    frac = (DWORD)(total * 0.005);
    cnt = 0;
    pEntry = &histogram[0];
    for ( j = 0; j < HISTOGRAM_SIZE; j++ ) {
        cnt += *pEntry++;
        if ( cnt > frac )
            break;
    }
    // if threshold is below minimum, leave black point set at 0
    if ( j < 10 )
        blackPt = 0;
    else if ( j > 100 ) {
        // threshold too high - find black 0.25% threshold point
        frac = (DWORD)(total*0.0025);
        cnt = 0;
        pEntry = &histogram[0];
        for ( k = 0; k < HISTOGRAM_SIZE; k++ ) {
            cnt += *pEntry++;
            if ( cnt > frac )
                break;
        }
        // if threshold still too high, use 100
        blackPt = ( k < 100 ) ? k : 100;
    } else
        blackPt = j;        // valid threshold found
```

```
    // find white 0.5% threshold point
    frac = (DWORD)(total * 0.005);
    cnt = 0;
    pEntry = &histogram[255];
    for ( j = HISTOGRAM_SIZE-1; j > 0; j-- ) {
        cnt += *pEntry--;
        if ( cnt > frac )
            break;
    }
    // if threshold is above maximum, leave white point set at 155
    if ( j > 245 )
        whitePt = 255;
    else if ( j < 155 ) {
        // threshold too low - find white 0.25% threshold point
        frac = (DWORD)(total*0.0025);
        cnt = 0;
        pEntry = &histogram[255];
        for ( k = HISTOGRAM_SIZE-1; k > 0; k-- ) {
            cnt += *pEntry--;
            if ( cnt > frac )
                break;
        }
        // if threshold still too low, use 155
        whitePt = ( k > 155 ) ? k : 155;
    } else
        whitePt = j;        // valid threshold found

}   // CPageTone :: CalculateWhiteBlackPoints


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: AutoAdjust
//
//  Purpose :   Generate auto-leveling curves to display in Automatic page
//
//---------------------------------------------------------------------------

void CPageTone :: AutoAdjust ( BOOL bDownload )
{
    CSpline     spline;
    int         whitePt, blackPt;
    CPoint      pts[2];
    DWORD       dwImageClass;

    // ignore tone adjust request unless image is in color
    g_Dispatch->GetCurrentCapabilitySetting ( CCAP_IMAGE_CLASS, &dwImageClass );
    if ( dwImageClass != IOP_IMAGECLASS_RGB )
        return;

    // force channel select to non-Master
    m_nChannel = CH_RED;

    // make sure application of tone curves is enabled
    g_Dispatch->EnableTone ( TRUE );

    // calculate histograms for all channels
    if ( g_pPicWnd->HasBitmap() ) {

        g_Dispatch->GetHistogram ( m_histogram[CH_MASTER], m_histogram[CH_RED], m_histogram
```

```
[CH_GREEN], m_histogram[CH_BLUE], &m_maxHistValue );

        // now, for all color channels
        for ( int channel = CH_RED; channel < NUM_CHANNELS; channel++ ) {
            // calculate white/black points
            CalculateWhiteBlackPoints ( m_histogram[channel], whitePt, blackPt );
            pts[0] = CPoint ( blackPt, 0 );
            pts[1] = CPoint ( whitePt, 255 );
            // compute curve
            spline.CalculateGraph ( m_aCurve[channel], pts, 2 );
            // download curve to IOP
            if ( bDownload ) {
                g_Dispatch->EnableRefresh ( channel == CH_BLUE );
                g_Dispatch->SetCurrentCapability ( ChannelToCap[channel], m_aCurve[channel]
);
            }
        }
    }

    InvalidateCurveWindow();

}   // CPageTone :: AutoAdjust


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: UpdateHistogram
//
//  Purpose :   If the histogram tone mode is active, update the histogram
//              information. Called whenever a new preview is done.
//
//---------------------------------------------------------------------------

void CPageTone :: UpdateHistogram()
{
    DWORD       dwImageClass;

    // ignore tone adjust request if image class is B&W or TextEnhance
    g_Dispatch->GetCurrentCapabilitySetting ( CCAP_IMAGE_CLASS, &dwImageClass );

    if ( dwImageClass == IOP_IMAGECLASS_BILEVEL || dwImageClass == IOP_IMAGECLASS_TRUST )
        return;

    if ( m_toneMode == IDC_PGTONE_HISTORADIO && g_pPicWnd->HasBitmap() ) {
        g_Dispatch->GetHistogram ( m_histogram[CH_MASTER], m_histogram[CH_RED], m_histogram
[CH_GREEN], m_histogram[CH_BLUE], &m_maxHistValue );
        InvalidateCurveWindow();
    }

}   // CPageTone :: UpdateHistogram


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: PostPendingError
//
//  Purpose :   Post message to self to provide delayed reporting of an error
//              associated with a particular field
//
//---------------------------------------------------------------------------
```

```
void CPageTone :: PostPendingError ( UINT strID, CString* pStr, CWnd& wnd )
{
    static CString  msg;

    if ( strID != 0 )
        msg.LoadString ( strID );
    else
        msg = *pStr;

    PostMessage ( WM_FIELDERROR, (WPARAM)&msg, (LPARAM)&wnd );

    // set ErrorPending flag to prevent Scan or Preview from proceeding
    m_bErrorPending = TRUE;
    CMainFrame* pMainWnd = (CMainFrame*)AfxGetMainWnd();
    CScuiView*  pView = (CScuiView*)pMainWnd->GetActiveView();
    pView->SetErrorPending ( TRUE );

}   // CPageTone :: PostPendingError


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: AdjustToneControls
//
//  Purpose :   Adjust tone control visibility due to change in color mode
//
//---------------------------------------------------------------------------

void CPageTone :: AdjustToneControls()
{
    if ( IsWindow ( m_hWnd ) )
        OnSelectToneMode ( m_toneMode );

}   // CPageTone :: AdjustToneControls


BEGIN_MESSAGE_MAP(CPageTone, CPropertyPage)
    //{{AFX_MSG_MAP(CPageTone)
    ON_WM_HSCROLL()
    ON_BN_CLICKED(IDC_PGTONE_RESETBUTTON, OnPgtoneResetbutton)
    ON_BN_CLICKED(IDC_PGTONE_LOADBUTTON, OnPgtoneLoadbutton)
    ON_BN_CLICKED(IDC_PGTONE_SAVEBUTTON, OnPgtoneSavebutton)
    ON_BN_CLICKED(IDC_PGTONE_HIGHLIGHTBUTTON, OnPgtoneHighlightbutton)
    ON_BN_CLICKED(IDC_PGTONE_MIDTONEBUTTON, OnPgtoneMidtonebutton)
    ON_BN_CLICKED(IDC_PGTONE_SHADOWBUTTON, OnPgtoneShadowbutton)
    ON_EN_KILLFOCUS(IDC_PGTONE_HIGHLIGHTEDIT, OnKillfocusPgtoneHighlightedit)
    ON_EN_KILLFOCUS(IDC_PGTONE_MIDTONEEDIT, OnKillfocusPgtoneMidtoneedit)
    ON_EN_KILLFOCUS(IDC_PGTONE_SHADOWEDIT, OnKillfocusPgtoneShadowedit)
    ON_CBN_SELCHANGE(IDC_PGTONE_PRESETS, OnSelchangePgtonePresets)
    ON_CBN_SELCHANGE(IDC_CHANNELCB, OnSelchangeChannelcb)
    ON_COMMAND_EX(IDC_PGTONE_AUTORADIO, OnSelectToneMode)
    ON_COMMAND_EX(IDC_PGTONE_BRICONTRADIO, OnSelectToneMode)
    ON_COMMAND_EX(IDC_PGTONE_CURVERADIO, OnSelectToneMode)
    ON_COMMAND_EX(IDC_PGTONE_GAMMARADIO, OnSelectToneMode)
    ON_COMMAND_EX(IDC_PGTONE_HISTORADIO, OnSelectToneMode)
    ON_EN_KILLFOCUS(IDC_PGTONE_GAMMAEDIT, OnKillfocusPgtoneGammaedit)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_CUSTOMCURVE, OnCustomCurve)
    ON_NOTIFY_EX(TTN_NEEDTEXT, 0, OnToolTipNotify)
```

```
    ON_MESSAGE(WM_FIELDERROR, OnFieldError)
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CPageTone message handlers

//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnInitDialog
//
//  Purpose  :   Initialize all of the page controls, load button bitmaps,
//               setup the tooltip control for this page
//
//---------------------------------------------------------------------------

BOOL CPageTone :: OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // subclass edit fields where we wish to process return and restrict to numbers
    m_shadowEdit.SubclassDlgItem ( IDC_PGTONE_SHADOWEDIT, this );
    m_midtoneEdit.SubclassDlgItem ( IDC_PGTONE_MIDTONEEDIT, this );
    m_hiliteEdit.SubclassDlgItem ( IDC_PGTONE_HIGHLIGHTEDIT, this );
    m_gammaEdit.SubclassDlgItem ( IDC_PGTONE_GAMMAEDIT, this );
    m_gammaEdit.SetFloat ( TRUE );

    LoadButtonBitmaps();
    LoadStaticBitmaps();

    // resize and reposition buttons
    CRect   btnRect;
    m_autoBtn.GetWindowRect ( btnRect );
    ScreenToClient ( btnRect );
    int     x = btnRect.left;
    int     y = btnRect.top;
    ResizeDialogButton ( m_autoBtn, x, y );     x += BUTTON_SIZE;
    ResizeDialogButton ( m_brightBtn, x, y );   x += BUTTON_SIZE;
    ResizeDialogButton ( m_gammaBtn, x, y );    x += BUTTON_SIZE;
    ResizeDialogButton ( m_histBtn, x, y );     x += BUTTON_SIZE;
    ResizeDialogButton ( m_customBtn, x, y );

    ResizeDialogButton ( m_shadowBtn, 0, 0, RM_OLD );
    ResizeDialogButton ( m_midtoneBtn, 0, 0, RM_OLD  );
    ResizeDialogButton ( m_hiliteBtn,  0, 0, RM_OLD );

    // get range of brightness. contrast and gamma
    S_IopCapRange<DWORD> irange;
    long pelement_count = 0;

    // get brightness range, and set slider limits
    g_Dispatch->GetFinalCapabilityRange( ICAP_BRIGHTNESS, &irange, &pelement_count);

    m_sliderBrightness.SetRange( (int)irange.min_value, (int)irange.max_value );
    m_sliderBrightness.SetTicFreq ( (int)((irange.max_value-irange.min_value)/10) );
    m_sliderBrightness.SetPageSize( (int)((irange.max_value-irange.min_value)/10) );

    // get contrast range, and set slider limits
    g_Dispatch->GetFinalCapabilityRange( ICAP_CONTRAST, &irange, &pelement_count);
```

```
m_sliderContrast.SetRange( (int)irange.min_value, (int)irange.max_value );
m_sliderContrast.SetTicFreq ( (int)((irange.max_value-irange.min_value)/10) );
m_sliderContrast.SetPageSize( (int)((irange.max_value-irange.min_value)/10) );

// get gamma range, and set slider limits
g_Dispatch->GetFinalCapabilityRange( ICAP_GAMMA, &irange, &pelement_count);

m_sliderGamma.SetRange( (int)irange.min_value, (int)irange.max_value );
m_sliderGamma.SetTicFreq ( (int)((irange.max_value-irange.min_value)/10) );
m_sliderGamma.SetPageSize( (int)((irange.max_value-irange.min_value)/10) );

// current brightness was retrieved during constructor, set slider position
m_sliderBrightness.SetPos( (int)m_nBright[CH_MASTER] );

// current contrast was retrieved during constructor, set slider position
m_sliderContrast.SetPos( (int)m_nContrast[CH_MASTER] );

// get current gamma value, and set slider position
g_Dispatch->GetCurrentCapabilitySetting(ICAP_GAMMA, &m_nGamma[CH_MASTER]);
m_sliderGamma.SetPos( (int)m_nGamma[CH_MASTER] );

// set all channel variables to the system defaults
for ( int chnl = 1; chnl < NUM_CHANNELS; chnl++ ) {
    m_nBright[chnl] = m_nBright[CH_MASTER];
    m_nContrast[chnl] = m_nContrast[CH_MASTER];
    m_nGamma[chnl] = m_nGamma[CH_MASTER];
}

m_presetCB.SetCurSel ( 0 );                    // default to Normal
m_channelCB.SetCurSel ( 0 );                   // default to Master channel

// initialize tooltip control, and add info for all of this page's controls
m_pToolTip = new CToolTipCtrl();
m_pToolTip->Create ( this );
m_pToolTip->AddTool ( &m_autoBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_brightBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_gammaBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_histBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_customBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_sliderBrightness, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_sliderContrast, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_sliderGamma, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_presetCB, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_shadowBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_midtoneBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_hiliteBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_loadBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_saveBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_resetBtn, LPSTR_TEXTCALLBACK );
m_pToolTip->AddTool ( &m_channelCB, LPSTR_TEXTCALLBACK );

m_pToolTip->Activate ( TRUE );

// create the curve editing window within the boundaries of the
// IDC_PAGETONE_CURVE static control. Static controls do not
// get keystrokes routed to them
CRect   curveRect;
CStatic*    pStatic = (CStatic*)GetDlgItem ( IDC_PGTONE_CURVE );
pStatic->GetWindowRect ( &curveRect );
ScreenToClient ( &curveRect );
```

```
    curveRect.InflateRect ( -2, -2 );
    m_pCurveWnd = new CCurveEditWnd;
    m_pCurveWnd->Create ( NULL, "CurveWnd", WS_CHILD, curveRect, this, IDC_CURVE_WINDOW );

    // create the gamma curve display window within the boundaries of the
    // IDC_PAGETONE_CURVE static control.
    m_pGammaWnd = new CGammaWnd;
    m_pGammaWnd->Create ( NULL, "GammaWnd", WS_CHILD | WS_VISIBLE, curveRect, this, IDC_GAM
MA_WINDOW );

    // create histogram control window
    CRect   ctrlRect;
    pStatic->GetWindowRect ( &ctrlRect );
    ctrlRect.top = ctrlRect.bottom + 2;
    ctrlRect.bottom += CONTROL_SIZE+4;
    ScreenToClient ( &ctrlRect );
    ctrlRect.left -= CONTROL_SIZE;
    ctrlRect.right += CONTROL_SIZE;
    m_pHistCtrls = new CHistogramControls;
    m_pHistCtrls->Create ( NULL, "HistCtrls", WS_CHILD, ctrlRect, this, IDC_HISTCTRL_WINDOW
);

    // initialize histogram controls
    SetDlgItemInt ( IDC_PGTONE_SHADOWEDIT, m_shadow[CH_MASTER] );
    SetDlgItemInt ( IDC_PGTONE_MIDTONEEDIT, m_midtone[CH_MASTER] );
    SetDlgItemInt ( IDC_PGTONE_HIGHLIGHTEDIT, m_highlight[CH_MASTER] );
    m_shadowEdit.SetLimitText ( 3 );
    m_midtoneEdit.SetLimitText ( 3 );
    m_hiliteEdit.SetLimitText ( 3 );

    // determine default starting page
    if ( m_bAuto ) {
        m_autoBtn.SetCheck ( 1 );
        OnSelectToneMode ( IDC_PGTONE_AUTORADIO );      // if Auto is ON, default to auto p
age
    } else {
        m_brightBtn.SetCheck ( 1 );                     // else default to contrast/brightn
ess page
        OnSelectToneMode( IDC_PGTONE_BRICONTRADIO );
    }

    return TRUE;

}   // CPageTone :: OnInitDialog


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnSysColorChange
//
//  Purpose :   Reloads and recolors all page bitmaps if system colors change
//
//---------------------------------------------------------------------------

void CPageTone :: OnSysColorChange()
{
    if ( ::IsWindow ( m_hWnd ) ) {
        LoadButtonBitmaps();
        LoadStaticBitmaps();
        //Invalidate();
```

```
    }

}   // CPageTone :: OnSysColorChange


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnSelectToneMode
//
//  Purpose  :   Show / hide all of the necessary controls, based on which
//               of the tone type buttons that has been pressed
//
//---------------------------------------------------------------------------

BOOL CPageTone :: OnSelectToneMode ( UINT nID )
{
    UINT    aHide[100];
    UINT    nHide = 0;
    UINT    aShow[30];
    UINT    nShow = 0;
    CString title;
    CRect   rect;
    CWaitCursor     cWait;
    CWnd*   pFocusWnd = NULL;
    DWORD   dwColor;
    static  int     nSaveChannel;

    // if returning from AutoTone, restore channel to what it was
    if ( m_toneMode == IDC_PGTONE_AUTORADIO ) {
        m_channelCB.SetCurSel ( nSaveChannel );
        OnSelchangeChannelcb();
    }

    m_toneMode = nID;

    aHide[nHide++] = IDC_PGTONE_CONTRASTSLIDER;
    aHide[nHide++] = IDC_PGTONE_CONTRASTPICLEFT;
    aHide[nHide++] = IDC_PGTONE_CONTRASTPICRIGHT;
    aHide[nHide++] = IDC_PGTONE_CONTRAST_LABEL;

    aHide[nHide++] = IDC_PGTONE_BRIGHTNESSSLIDER;
    aHide[nHide++] = IDC_PGTONE_BRIGHTNESSPICLEFT;
    aHide[nHide++] = IDC_PGTONE_BRIGHTNESSPICRIGHT;
    aHide[nHide++] = IDC_PGTONE_BRIGHT_LABEL;

    aHide[nHide++] = IDC_PGTONE_GAMMASLIDER;
    aHide[nHide++] = IDC_PGTONE_GAMMAPICLEFT;
    aHide[nHide++] = IDC_PGTONE_GAMMAPICRIGHT;
    aHide[nHide++] = IDC_PGTONE_GAMMA_LABEL;
    aHide[nHide++] = IDC_PGTONE_GAMMAEDIT;

    aHide[nHide++] = IDC_PGTONE_SHADOWEDIT;
    aHide[nHide++] = IDC_PGTONE_SHADOWBUTTON;
    aHide[nHide++] = IDC_PGTONE_SHADOWLABEL;
    aHide[nHide++] = IDC_PGTONE_MIDTONEEDIT;
    aHide[nHide++] = IDC_PGTONE_MIDTONEBUTTON;
    aHide[nHide++] = IDC_PGTONE_MIDTONELABEL;
    aHide[nHide++] = IDC_PGTONE_HIGHLIGHTEDIT;
    aHide[nHide++] = IDC_PGTONE_HIGHLIGHTBUTTON;
    aHide[nHide++] = IDC_PGTONE_HIGHLIGHTLABEL;
```

```
aHide[nHide++] = IDC_PGTONE_LOADBUTTON;
aHide[nHide++] = IDC_PGTONE_SAVEBUTTON;
aHide[nHide++] = IDC_PGTONE_PRESETS;
aHide[nHide++] = IDC_PGTONE_SPECIAL_LABEL;

for ( UINT i = 0 ; i < nHide ; ++i )
    GetDlgItem( aHide[i] )->ShowWindow( SW_HIDE );

switch ( nID ) {

case IDC_PGTONE_AUTORADIO:
    title.LoadString ( IDS_AUTO );
    SetDlgItemText ( IDC_PGTONE_TITLE, title );
    // force channel select away from Master
    nSaveChannel = m_nChannel;
    if ( m_nChannel == CH_MASTER ) {
        m_channelCB.SetCurSel ( CH_RED );
        OnSelchangeChannelcb();
    }
    AutoAdjust(TRUE);
    m_pGammaWnd->ShowWindow ( SW_SHOW );
    m_pCurveWnd->ShowWindow ( SW_HIDE );
    m_pHistCtrls->ShowWindow ( SW_HIDE );
    GetDlgItem ( IDC_PGTONE_RESETBUTTON )->ShowWindow ( SW_HIDE );
    break;

case IDC_PGTONE_BRICONTRADIO:
    title.LoadString ( IDS_CONTRAST );
    SetDlgItemText ( IDC_PGTONE_TITLE, title );
    aShow[nShow++] = IDC_PGTONE_CONTRASTSLIDER;
    aShow[nShow++] = IDC_PGTONE_CONTRASTPICLEFT;
    aShow[nShow++] = IDC_PGTONE_CONTRASTPICRIGHT;
    aShow[nShow++] = IDC_PGTONE_CONTRAST_LABEL;

    aShow[nShow++] = IDC_PGTONE_BRIGHTNESSSLIDER;
    aShow[nShow++] = IDC_PGTONE_BRIGHTNESSPICLEFT;
    aShow[nShow++] = IDC_PGTONE_BRIGHTNESSPICRIGHT;
    aShow[nShow++] = IDC_PGTONE_BRIGHT_LABEL;
    aShow[nShow++] = IDC_PGTONE_RESETBUTTON;
    m_pGammaWnd->ShowWindow ( SW_SHOW );
    m_pCurveWnd->ShowWindow ( SW_HIDE );
    m_pHistCtrls->ShowWindow ( SW_HIDE );
    break;

case IDC_PGTONE_GAMMARADIO:
    title.LoadString ( IDS_GAMMA );
    SetDlgItemText ( IDC_PGTONE_TITLE, title );
    aShow[nShow++] = IDC_PGTONE_GAMMASLIDER;
    aShow[nShow++] = IDC_PGTONE_GAMMAPICLEFT;
    aShow[nShow++] = IDC_PGTONE_GAMMAPICRIGHT;
    aShow[nShow++] = IDC_PGTONE_GAMMAEDIT;
    aShow[nShow++] = IDC_PGTONE_GAMMA_LABEL;
    aShow[nShow++] = IDC_PGTONE_RESETBUTTON;
    m_pGammaWnd->ShowWindow ( SW_SHOW );
    m_pCurveWnd->ShowWindow ( SW_HIDE );
    m_pHistCtrls->ShowWindow ( SW_HIDE );
    pFocusWnd = GetDlgItem ( IDC_PGTONE_GAMMAEDIT );
    break;

case IDC_PGTONE_HISTORADIO: {
```

```
        title.LoadString ( IDS_HISTOGRAM );
        SetDlgItemText ( IDC_PGTONE_TITLE, title );
        aShow[nShow++] = IDC_PGTONE_SHADOWEDIT;
        aShow[nShow++] = IDC_PGTONE_SHADOWBUTTON;
        aShow[nShow++] = IDC_PGTONE_SHADOWLABEL;
        aShow[nShow++] = IDC_PGTONE_MIDTONEEDIT;
        aShow[nShow++] = IDC_PGTONE_MIDTONEBUTTON;
        aShow[nShow++] = IDC_PGTONE_MIDTONELABEL;
        aShow[nShow++] = IDC_PGTONE_HIGHLIGHTEDIT;
        aShow[nShow++] = IDC_PGTONE_HIGHLIGHTBUTTON;
        aShow[nShow++] = IDC_PGTONE_HIGHLIGHTLABEL;
        aShow[nShow++] = IDC_PGTONE_LOADBUTTON;
        aShow[nShow++] = IDC_PGTONE_SAVEBUTTON;
        aShow[nShow++] = IDC_PGTONE_RESETBUTTON;
        m_pGammaWnd->ShowWindow ( SW_SHOW );
        m_pCurveWnd->ShowWindow ( SW_HIDE );
        m_pHistCtrls->ShowWindow ( SW_SHOW );
        UpdateWindow();
        // get histogram information for current image
        if ( g_pPicWnd->HasBitmap() )
            g_Dispatch->GetHistogram ( m_histogram[CH_MASTER], m_histogram[CH_RED], m_histo
gram[CH_GREEN], m_histogram[CH_BLUE], &m_maxHistValue );
        pFocusWnd = GetDlgItem ( IDC_PGTONE_SHADOWEDIT );
    }   break;

    case IDC_PGTONE_CURVERADIO: {
        title.LoadString ( IDS_CURVE );
        SetDlgItemText ( IDC_PGTONE_TITLE, title );
        aShow[nShow++] = IDC_PGTONE_PRESETS;
        aShow[nShow++] = IDC_PGTONE_LOADBUTTON;
        aShow[nShow++] = IDC_PGTONE_SAVEBUTTON;
        aShow[nShow++] = IDC_PGTONE_RESETBUTTON;
        aShow[nShow++] = IDC_PGTONE_SPECIAL_LABEL;
        m_pGammaWnd->ShowWindow ( SW_HIDE );
        m_pCurveWnd->ShowWindow ( SW_SHOW );
        m_pHistCtrls->ShowWindow ( SW_HIDE );
    }   break;

    default:
        ASSERT( FALSE );
        break;
    }

    m_bAuto = nID == IDC_PGTONE_AUTORADIO;

    // make sure that the Auto button on the toolbar matches our current auto mode
    CMainFrame* pMainFrame = (CMainFrame*)AfxGetMainWnd();
    pMainFrame->SetAutoLevel ( m_bAuto );

    for ( i = 0 ; i < nShow ; ++i )
        GetDlgItem( aShow[i] )->ShowWindow( SW_SHOWNA );

    // the method above of turning everything off, then back on, causes items
    // that are common between the tone pages to blink. therefore, the following
    // controls that are present for almost all of the pages will be
    // handled differently
    int     iShow = ( nID == IDC_PGTONE_HISTORADIO ) ? SW_HIDE : SW_SHOWNA;
    GetDlgItem ( IDC_WHITE1 )->ShowWindow ( iShow );
    GetDlgItem ( IDC_WHITE2 )->ShowWindow ( iShow );
    GetDlgItem ( IDC_BLACK  )->ShowWindow ( iShow );
```

```
    GetDlgItem ( IDC_GRAY1  )->ShowWindow ( iShow );
    GetDlgItem ( IDC_GRAY2  )->ShowWindow ( iShow );

    g_Dispatch->GetCurrentCapabilitySetting ( CCAP_IMAGE_CLASS, &dwColor );
    iShow = ( nID == IDC_PGTONE_AUTORADIO || dwColor == IOP_IMAGECLASS_GRAY) ? SW_HIDE : SW
_SHOWNA;
    GetDlgItem ( IDC_CHANNELCB )->ShowWindow ( iShow );
    GetDlgItem ( IDC_PGTONE_CHLABEL )->ShowWindow ( iShow );

    // if Grayscale mode, then force channel to Master
    if ( dwColor == IOP_IMAGECLASS_GRAY ) {
        m_channelCB.SetCurSel ( CH_MASTER );
        OnSelchangeChannelcb();
    }

    UpdateControls();
    UpdateWindow();
    InvalidateCurveWindow();

    DownloadAllCurves();

    // force focus to the appropriate control for this mode
    if ( pFocusWnd )
        pFocusWnd->SetFocus();

    return TRUE;

}   // CPageTone :: OnSelectToneMode


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnHScroll
//
//  Purpose :   Process changes in the tone control caused by the user altering
//              the settings of one of the horizontal slider controls
//
//---------------------------------------------------------------------------

void CPageTone :: OnHScroll ( UINT nSBCode, UINT /*nPos*/, CScrollBar* pScrollBar )
{
    // first check if we need to update the gamma edit field on the fly, before release
    if ( nSBCode == TB_THUMBTRACK && pScrollBar->GetSafeHwnd() == m_sliderGamma.m_hWnd ) {
            m_nGamma[m_nChannel] = m_sliderGamma.GetPos();

            double      dGamma;
            if ( m_nGamma[m_nChannel] <= 50 )
                dGamma = 0.018 * m_nGamma[m_nChannel] + 0.1;    // convert (0-50) to (0.1-1)
            else
                dGamma = 0.18 * m_nGamma[m_nChannel] - 8;       // convert (50-100) to (1-10
)
            CString str;
            str.Format ( "%3.1f", dGamma );
            SetDlgItemText ( IDC_PGTONE_GAMMAEDIT, str );
    }

    // in all cases, the picture is not updated until the user ends the scroll movement
    if ( nSBCode != TB_ENDTRACK && nSBCode != TB_THUMBTRACK ) {

        // check for the gamma slider
```

```
    if ( pScrollBar->GetSafeHwnd() == m_sliderGamma.m_hWnd )    {
        m_nGamma[m_nChannel] = m_sliderGamma.GetPos();

        double      dGamma;
        if ( m_nGamma[m_nChannel] <= 50 )
            dGamma = 0.018 * m_nGamma[m_nChannel] + 0.1;    // convert (0-50) to (0.1-1)
        else
            dGamma = 0.18 * m_nGamma[m_nChannel] - 8;       // convert (50-100) to (1-10
)
        CString str;
        str.Format ( "%3.1f", dGamma );
        SetDlgItemText ( IDC_PGTONE_GAMMAEDIT, str );
        ImageIF.CurveFromGamma ( m_aCurve[m_nChannel], dGamma );
        DownloadCurve ( m_nChannel );

    } else {
        // must be the brightness or contrast slider
        ASSERT( pScrollBar->GetSafeHwnd() == m_sliderBrightness.m_hWnd ||
                pScrollBar->GetSafeHwnd() == m_sliderContrast.m_hWnd );
        m_nBright[m_nChannel] = m_sliderBrightness.GetPos();
        m_nContrast[m_nChannel]   = m_sliderContrast.GetPos();

        ImageIF.CurveFromBC ( m_aCurve[m_nChannel], m_nBright[m_nChannel]*2-100, m_nCon
trast[m_nChannel]*2-100 );
        DownloadCurve ( m_nChannel );
    }

    InvalidateCurveWindow();
}

}   // CPageTone :: OnHScroll


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnPgtoneResetbutton
//
//  Purpose :   Reset whichever tone controls are currently active to their
//              default settings, and show the new default curve
//
//---------------------------------------------------------------------------

void CPageTone :: OnPgtoneResetbutton()
{
    CRect       rect;
    CWaitCursor wait;

    switch ( m_toneMode ) {

    case IDC_PGTONE_BRICONTRADIO:
        // reset brightness and contrast controls to default values
        g_Dispatch->GetDefaultCapabilitySetting ( ICAP_BRIGHTNESS, &m_nBright[m_nChannel] )
;
        m_sliderBrightness.SetPos ( (int)m_nBright[m_nChannel] );
        g_Dispatch->GetDefaultCapabilitySetting ( ICAP_CONTRAST, &m_nContrast[m_nChannel] )
;
        m_sliderContrast.SetPos ( (int)m_nContrast[m_nChannel] );
        ImageIF.CurveFromBC ( m_aCurve[m_nChannel], m_nBright[m_nChannel]*2-100, m_nContras
t[m_nChannel]*2-100 );
        g_Dispatch->SetCurrentCapability ( ChannelToCap[m_nChannel], m_aCurve[m_nChannel] )
```

```
        break;

    case IDC_PGTONE_GAMMARADIO: {
        // reset gamma control to default value
        g_Dispatch->GetDefaultCapabilitySetting ( ICAP_GAMMA, &m_nGamma[m_nChannel] );
        m_sliderGamma.SetPos ( (int)m_nGamma[m_nChannel] );
        double      dGamma;
        if ( m_nGamma[m_nChannel] <= 50 )
            dGamma = 0.018 * m_nGamma[m_nChannel] + 0.1;    // convert (0-50) to (0.1-1)
        else
            dGamma = 0.18 * m_nGamma[m_nChannel] - 8;       // convert (50-100) to (1-10)
        CString str;
        str.Format ( "%3.1f", dGamma );
        SetDlgItemText ( IDC_PGTONE_GAMMAEDIT, str );
        ImageIF.CurveFromGamma ( m_aCurve[m_nChannel], dGamma );
        g_Dispatch->SetCurrentCapability ( ChannelToCap[m_nChannel], m_aCurve[m_nChannel] )
;
    }   break;

    case IDC_PGTONE_HISTORADIO:
        // reset shadow, gray and white points to produce normal curve
        SetShadow ( 0 );        m_pHistCtrls->SetShadow ( 0 );
        SetHighlight ( 255 );   m_pHistCtrls->SetHighlight ( 255 );
        SetMidtone ( 128 );     m_pHistCtrls->SetMidtone ( 128, TRUE );
        ComputeHistoCurve ( m_nChannel );
        break;

    case IDC_PGTONE_CURVERADIO:
        // set dropdown to custom
        m_presetCB.SetCurSel ( 5 );
        OnSelchangePgtonePresets();
        break;
    }

    // now get new values for curve, and force a repaint of the curve window
    // g_Dispatch->GetCurrentCapabilitySetting ( ChannelToCap[m_nChannel], m_aCurve[m_nChan
nel] );

    InvalidateCurveWindow();

}   // CPageTone :: OnPgtoneResetbutton


//-----------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnToolTipNotify
//
//  Purpose :   Fetches the tooltip string associated with the specified
//              window, and updates the status line with explanatory text
//
//-----------------------------------------------------------------------

BOOL CPageTone :: OnToolTipNotify ( UINT /*id*/, NMHDR* pNMHDR, LRESULT* /*pResult*/ )
{
    return ::OnToolTipNotify ( pNMHDR );

}   // CPageTone :: OnToolTipNotify
```

```
//-----------------------------------------------------------------------
//
//  Method   :   CPageTone :: PreTranslateMessage
//
//  Purpose  :   All mouse messages must be intercepted and fed to this page's
//               tooltip control if tooltips are to be processed properly
//
//-----------------------------------------------------------------------

BOOL CPageTone :: PreTranslateMessage ( MSG* pMsg )
{
    switch ( pMsg->message ) {

        case WM_MOUSEMOVE:
        case WM_LBUTTONDOWN:
        case WM_LBUTTONUP:
        case WM_MBUTTONDOWN:
        case WM_MBUTTONUP:
        case WM_RBUTTONDOWN:
        case WM_RBUTTONUP:
            if ( m_pToolTip != NULL ) {
                // this will force reactivation of the tooltip, which is
                // sometimes disabled by MFC's dialog handling
                m_pToolTip->Activate ( TRUE );
                m_pToolTip->RelayEvent ( pMsg );
            }
            break;
    }

    return CPropertyPage::PreTranslateMessage(pMsg);

}   // CPageTone :: PreTranslateMessage


//-----------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnSelchangePgtonePresets
//
//  Purpose  :   Set the curve points to reflect the selected entry from
//               the preset adjustments combo box, redraw the curve and
//               download the new curve array to IOP
//
//-----------------------------------------------------------------------

void CPageTone :: OnSelchangePgtonePresets()
{
    int         channel;
    int         blackPt, whitePt;
    CWaitCursor wait;

    m_pCurveWnd->Reset ( m_nChannel );

    switch ( m_presetCB.GetCurSel() ) {

    case 0 : // Normal
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 0, 0 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 255, 255 ) );
        break;

    case 1 : // Underexposed
```

```
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 0, 0 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 30, 103 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 100, 195 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 255, 255 ) );
        break;

    case 2 : // Overexposed
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 0, 0 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 155, 131 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 255, 255 ) );
        break;

    case 3 : // Low Contrast
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 0, 0 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 75, 40 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 175, 225 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 255, 255 ) );
        break;

    case 4:  // Automatic
        if ( g_pPicWnd->HasBitmap() ) {
            // calculate histograms for all channels
            g_Dispatch->GetHistogram ( m_histogram[CH_MASTER], m_histogram[CH_RED], m_histo
gram[CH_GREEN], m_histogram[CH_BLUE], &m_maxHistValue );
            // now, for all channels
            for ( channel = CH_MASTER; channel < NUM_CHANNELS; channel++ ) {
                m_pCurveWnd->Reset ( channel );
                // calculate white/black points
                CalculateWhiteBlackPoints ( m_histogram[channel], whitePt, blackPt );
                // add points to curve
                m_pCurveWnd->AddPoint ( channel, CPoint ( blackPt, 0 ) );
                m_pCurveWnd->AddPoint ( channel, CPoint ( whitePt, 255 ) );
                // compute curve
                m_pCurveWnd->GenerateCurve ( channel );
                // download curve to IOP
                g_Dispatch->EnableRefresh ( channel == CH_BLUE );
                g_Dispatch->SetCurrentCapability ( ChannelToCap[channel], m_pCurveWnd->GetG
raphData(channel) );
            }
        }
        return;
        break;

    case 5 : // Custom
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 0, 0 ) );
        m_pCurveWnd->AddPoint ( m_nChannel, CPoint ( 255, 255 ) );
        break;
    }

    m_bAuto = m_presetCB.GetCurSel() == 4;

    m_pCurveWnd->GenerateCurve ( m_nChannel );

    // download new curve to IOP
    g_Dispatch->SetCurrentCapability ( ChannelToCap[m_nChannel], m_pCurveWnd->GetGraphData(
m_nChannel) );

}   // CPageTone :: OnSelchangePgtonePresets
```

```
//----------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnPgtoneLoadbutton
//
//  Purpose  :   Brings up system Open File dialog box, and if successful,
//               invokes the method necessary to load a set of custom curve
//               points from the selected disk file
//
//----------------------------------------------------------------------

void CPageTone :: OnPgtoneLoadbutton()
{
    CString         fileType;
    char*           pFilePath;

    ScannerInterface*   pScanIntf = g_Dispatch->GetDeviceObject();
    pFilePath = pScanIntf->GetDSFileName();
    CString     filePath ( pFilePath );

    int index = filePath.ReverseFind ( '\\' );
    if ( index != -1 )
        filePath.SetAt ( index+1, 0 );

    switch ( m_toneMode ) {

        case IDC_PGTONE_HISTORADIO: {
            fileType.LoadString ( IDS_HISTOFILES );

            CFileDialog     fileDlg ( TRUE, "hst", "*.hst", OFN_HIDEREADONLY | OFN_OVERWRIT
EPROMPT, fileType );

            fileDlg.m_ofn.lpstrInitialDir = filePath;

            if ( fileDlg.DoModal() == IDOK ) {

                // have the curve editing window load the set of user-defined curve points
                CString     fileName = fileDlg.GetPathName();
                m_pGammaWnd->LoadCurve ( fileName );
                UpdateControls();

            }
        }   break;

        case IDC_PGTONE_CURVERADIO: {
            fileType.LoadString ( IDS_CURVEFILES );

            CFileDialog     fileDlg ( TRUE, "crv", "*.crv", OFN_HIDEREADONLY | OFN_OVERWRIT
EPROMPT, fileType );

            fileDlg.m_ofn.lpstrInitialDir = filePath;

            if ( fileDlg.DoModal() == IDOK ) {

                // have the curve editing window load the set of user-defined curve points
                CString     fileName = fileDlg.GetPathName();
                m_pCurveWnd->LoadCurve ( fileName );
                OnCustomCurve ( 0, 0 );

            }
        }   break;
```

```
    }
}   // CPageTone :: OnPgtoneLoadbutton


//----------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnPgtoneSaveButton
//
//  Purpose  :   Brings up system Save As dialog box, and if successful,
//               invokes the method necessary to save the current custom
//               curve points to the selected disk file
//
//----------------------------------------------------------------------

void CPageTone :: OnPgtoneSavebutton()
{
    CString         fileType;
    char*           pFilePath;

    ScannerInterface*   pScanIntf = g_Dispatch->GetDeviceObject();
    pFilePath = pScanIntf->GetDSFileName();
    CString     filePath ( pFilePath );

    int index = filePath.ReverseFind ( '\\' );
    if ( index != -1 )
        filePath.SetAt ( index+1, 0 );

    switch ( m_toneMode ) {

        case IDC_PGTONE_HISTORADIO: {
            fileType.LoadString ( IDS_HISTOFILES );

            CFileDialog     fileDlg ( FALSE, "hst", "*.hst", OFN_HIDEREADONLY | OFN_OVERWRI
TEPROMPT, fileType );

            fileDlg.m_ofn.lpstrInitialDir = filePath;

            if ( fileDlg.DoModal() == IDOK ) {

                // have curve editing window save the user-defined curve points
                CString     fileName = fileDlg.GetPathName();
                m_pGammaWnd->SaveCurve ( fileName );

            }
        }   break;

        case IDC_PGTONE_CURVERADIO: {
            fileType.LoadString ( IDS_CURVEFILES );

            CFileDialog     fileDlg ( FALSE, "crv", "*.crv", OFN_HIDEREADONLY | OFN_OVERWRI
TEPROMPT, fileType );

            fileDlg.m_ofn.lpstrInitialDir = filePath;

            if ( fileDlg.DoModal() == IDOK ) {

                // have curve editing window save the user-defined curve points
                CString     fileName = fileDlg.GetPathName();
                m_pCurveWnd->SaveCurve ( fileName );
```

```
            }
        }   break;
    }

}   // CPageTone :: OnPgtoneSaveButton


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnCustomCurve
//
//  Purpose :   Whenever user clicks within the curve editing window, we
//              must change the value of the preset combobox to Custom.
//
//---------------------------------------------------------------------------

LRESULT CPageTone :: OnCustomCurve ( WPARAM /*wParam*/, LPARAM /*lParam*/ )
{
    // window is displaying a custom curve; set combobox to custom
    m_presetCB.SetCurSel ( 5 );
    m_bAuto = FALSE;

    return 0;

}   // CPageTone :: OnCustomCurve


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnPgtoneHighlightbutton
//
//  Purpose :   Select highlight value by using eyedropper on image window
//
//---------------------------------------------------------------------------

void CPageTone :: OnPgtoneHighlightbutton()
{
    CMainFrame* pMainFrame = (CMainFrame*)AfxGetMainWnd();

    pMainFrame->SetUIMode ( CScuiDispatch :: uimodeHilite );

}   // CPageTone :: OnPgtoneHighlightbutton


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnPgtoneMidtonebutton
//
//  Purpose :   Select midtone value by using eyedropper on image window
//
//---------------------------------------------------------------------------

void CPageTone :: OnPgtoneMidtonebutton()
{
    CMainFrame* pMainFrame = (CMainFrame*)AfxGetMainWnd();

    pMainFrame->SetUIMode ( CScuiDispatch :: uimodeMidtone );
```

```
}   // CPageTone :: OnPgtoneMidtonebutton


//-----------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnPgtoneShadowbutton
//
//  Purpose  :   Select shadow value by using eyedropper on image window
//
//-----------------------------------------------------------------------

void CPageTone :: OnPgtoneShadowbutton()
{
    CMainFrame* pMainFrame = (CMainFrame*)AfxGetMainWnd();

    pMainFrame->SetUIMode ( CScuiDispatch :: uimodeShadow );


}   // CPageTone :: OnPgtoneShadowbutton


//-----------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnKillfocusPgtoneHighlightedit
//
//  Purpose  :   Accept user entry for highlight value
//
//-----------------------------------------------------------------------

void CPageTone::OnKillfocusPgtoneHighlightedit()
{
    int value = GetDlgItemInt ( IDC_PGTONE_HIGHLIGHTEDIT );

    if ( value > 255 ) {
        PostPendingError ( IDS_INVTONEVALUE, NULL, m_hiliteEdit );
        return;
    }

    m_highlight[m_nChannel] = (BYTE)value;
    m_pHistCtrls->SetHighlight ( m_highlight[m_nChannel] );
    ComputeHistoCurve ( m_nChannel );

}   // CPageTone :: OnKillfocusPgtoneHighlightedit


//-----------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnKillfocusPgtoneMidtoneedit
//
//  Purpose  :   Accept user entry for midtone value
//
//-----------------------------------------------------------------------

void CPageTone::OnKillfocusPgtoneMidtoneedit()
{
    int value = GetDlgItemInt ( IDC_PGTONE_MIDTONEEDIT );
    int oldValue = value;

    if ( value > 255 ) {
        PostPendingError ( IDS_INVTONEVALUE, NULL, m_midtoneEdit );
```

```
        return;
    }

    if ( m_shadow[m_nChannel] < m_highlight[m_nChannel] ) {
        if ( value < m_shadow[m_nChannel] )
            value = m_shadow[m_nChannel];
        else if ( value > m_highlight[m_nChannel] )
            value = m_highlight[m_nChannel];
    } else {
        if ( value < m_highlight[m_nChannel] )
            value = m_highlight[m_nChannel];
        else if ( value > m_shadow[m_nChannel] )
            value = m_shadow[m_nChannel];
    }

    if ( value != oldValue ) {
        SetDlgItemInt ( IDC_PGTONE_MIDTONEEDIT, value, FALSE );
    }

    if ( m_midtone[m_nChannel] != value ) {
        m_midtone[m_nChannel] = (BYTE)value;
        m_pHistCtrls->SetMidtone ( m_midtone[m_nChannel], TRUE );
        ComputeHistoCurve ( m_nChannel );
    }

}   // CPageTone :: OnKillfocusPgtoneMidtoneedit


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnKillfocusPgtoneShadowedit
//
//  Purpose :   Accept user entry for shadow value
//
//---------------------------------------------------------------------------

void CPageTone::OnKillfocusPgtoneShadowedit()
{
    int value = GetDlgItemInt ( IDC_PGTONE_SHADOWEDIT );

    if ( value > 255 ) {
        PostPendingError ( IDS_INVTONEVALUE, NULL, m_shadowEdit );
        return;
    }

    m_shadow[m_nChannel] = (BYTE)value;
    m_pHistCtrls->SetShadow ( m_shadow[m_nChannel] );
    ComputeHistoCurve ( m_nChannel );

}   // CPageTone :: OnKillfocusPgtoneShadowedit


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnSelchangeChannelcb
//
//  Purpose :   Handle user channel change
//
//---------------------------------------------------------------------------
```

```
void CPageTone::OnSelchangeChannelcb()
{
    m_nChannel = m_channelCB.GetCurSel();

    // update all controls to reflect new channel
    UpdateControls();

    // force download of this channel, so that IOP knows whether we are
    // looking at the MASTER channel or one of the RGB channels
    RecomputeCurve ( m_nChannel );

}   // CPageTone :; OnSelchangeChannelcb


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnKillfocusPgtoneGammaedit
//
//  Purpose :   Handle user channel change
//
//---------------------------------------------------------------------------

void CPageTone::OnKillfocusPgtoneGammaedit()
{
    // get text string
    CString     str;
    GetDlgItemText ( IDC_PGTONE_GAMMAEDIT, str );

    // convert to double value
    double      dGamma = atof ( str );
    if ( dGamma < 0.1 || dGamma > 10.0 ) {
        PostPendingError ( IDS_INVALID_GAMMA, NULL, m_gammaEdit );
        return;
    }

    // put number back into edit field in standard format (force inclusion of decimal pt)
    str.Format ( "%3.1f", dGamma );
    SetDlgItemText ( IDC_PGTONE_GAMMAEDIT, str );

    // convert this to integer control setting
    int         nPos;
    if ( dGamma <= 1.0 )
        nPos = (int)(( dGamma - 0.1 ) / 0.018);             // convert (0.1-1) to (0-50)
    else
        nPos = (int)(( dGamma + 8.0 ) / 0.18);              // convert (1-10) to (50-100)

    // update control
    m_sliderGamma.SetPos( nPos );

    // create new curve and update IOP
    ImageIF.CurveFromGamma ( m_aCurve[m_nChannel], dGamma );
    g_Dispatch->SetCurrentCapability ( ChannelToCap[m_nChannel], m_aCurve[m_nChannel] );
    InvalidateCurveWindow();

}   // CPageTone :: OnKillfocusPgtoneGammaedit


//---------------------------------------------------------------------------
//
//  Method  :   CPageTone :: SetAutoLevel
```

```
//
//  Purpose :   Change the status of auto-leveling, and insure the correct
//              tone page is being shown
//
//-----------------------------------------------------------------------

void CPageTone :: SetAutoLevel ( BOOL bAuto )
{
    m_bAuto = bAuto;

    // if this page has already been created, insure that the proper tone mode is selected
    if ( GetSafeHwnd() != NULL ) {

        if ( m_bAuto ) {
            m_autoBtn.SetCheck ( 1 );
            m_brightBtn.SetCheck ( 0 );
            m_gammaBtn.SetCheck ( 0 );
            m_histBtn.SetCheck ( 0 );
            m_customBtn.SetCheck ( 0 );
            OnSelectToneMode ( IDC_PGTONE_AUTORADIO );      // if Auto is ON, default to au
to page
        } else {
            if ( m_autoBtn.GetCheck() == 1 ) {              // if auto turned off,
                m_autoBtn.SetCheck ( 0 );
                m_brightBtn.SetCheck ( 1 );                 // default to contrast/brightne
ss page
                OnSelectToneMode( IDC_PGTONE_BRICONTRADIO );
            }
        }
    }

}   // CPageTone :: SetAutoLevel


//-----------------------------------------------------------------------
//
//  Method  :   CPageTone :: OnFieldError
//
//  Purpose :   Post requested error message, and send focus back to control.
//              This is used whenever data is validated within the KillFocus
//              handler of a control, since it is not possible to redirect
//              the focus back to the control while MFC is trying to change
//              the focus. So a WM_FIELDERROR message is posted, and this
//              routine later handles the error message and focus change.
//
//-----------------------------------------------------------------------

LRESULT CPageTone :: OnFieldError ( WPARAM wParam, LPARAM lParam )
{
    CString*    pMsg = (CString*)wParam;
    CWnd*       pWnd = (CWnd*)lParam;

    AfxMessageBox ( *pMsg, MB_OK | MB_ICONSTOP );
    GotoDlgCtrl ( pWnd );

    // clear ErrorPending flag which prevents Scan or Preview from being
    // performed when a data validation error occurs
    CMainFrame* pMainWnd = (CMainFrame*)AfxGetMainWnd();
    CScuiView*  pView = (CScuiView*)pMainWnd->GetActiveView();
    pView->SetErrorPending ( FALSE );
```

```
    return 0;

}   // CPageTone :: OnFieldError


//---------------------------------------------------------------------------
//
//  Method   :   CPageTone :: OnKillActive
//
//  Purpose  :   Prevents leaving the current page if any field has reported
//               a data validation error.
//
//  Returns  :   FALSE if m_bErrorPending is set
//
//---------------------------------------------------------------------------

BOOL CPageTone :: OnKillActive()
{
    // force the focus to the channel combo, simply to force OnKillFocus to be
    // called for any field that does its validation there
    m_channelCB.SetFocus();

    if ( m_bErrorPending ) {
        m_bErrorPending = FALSE;
        return FALSE;
    }

    return CPropertyPage::OnKillActive();

}   // CPageTone :: OnKillActive
```

Appendix B

Object Script "pre" ID = 114
Friday, September 26, 1997 7:19 PM

```
on mouseUp
  set visible of graphic "pre on" to true
  set visible of graphic "gama on" to false
  set visible of graphic "use tone" to false
  set visible of graphic "levels on" of card "tone" to false
  set visible of graphic "curves on" of card "tone" to false
  set visible of graphic "gama on 2" of card "tone" to false
  set visible of graphic "pre on 2" of card "tone" to true
end mouseUp

on mouseEnter
  set visible of graphic "pre label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "pre label" to false
end mouseLeave
```

Object Script "gama" ID = 107
Friday, September 26, 1997 7:19 PM

```
on mouseUp
  set visible of graphic "gama on" to true
  set visible of graphic "pre on" to false
  set visible of graphic "use tone" to false
  set visible of graphic "levels on" of card "tone" to false
  set visible of graphic "curves on" of card "tone" to false
  set visible of graphic "gama on 2" of card "tone" to true
  set visible of graphic "pre on 2" of card "tone" to false
end mouseUp

on mouseEnter
  set visible of graphic "gama label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "gama label" to false
end mouseLeave
```

Object Script "BC" ID = 103
Friday, September 26, 1997 7:18 PM

```
on mouseUp
  set visible of graphic "gama on" to false
  set visible of graphic "pre on" to false
  set visible of graphic "use tone" to false
  set visible of graphic "levels on" of card "tone" to false
  set visible of graphic "curves on" of card "tone" to false
  set visible of graphic "gama on 2" of card "tone" to false
  set visible of graphic "pre on 2" of card "tone" to false
end mouseUp

on mouseEnter
  set visible of graphic "BC label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "BC label" to false
end mouseLeave
```

Object Script "pre" ID = 162
Friday, September 26, 1997 7:18 PM

```
on mouseUp
  set visible of graphic "curves on" to true
  set visible of graphic "pre on 2" to false
  set visible of graphic "gama on 2" to false
  set visible of graphic "levels on" to false  set visible of graphic "use tone" of card
  set visible of graphic "gama on" of card "main" to false
  set visible of graphic "pre on" of card "main" to false
end mouseUp

on mouseEnter
  set visible of graphic "curves label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "curves label" to false
end mouseLeave
```

Object Script "pre" ID = 114
Friday, September 26, 1997 7:18 PM

```
on mouseUp
  set visible of graphic "pre on 2" to true
  set visible of graphic "gama on 2" to false
  set visible of graphic "levels on" to false
  set visible of graphic "curves on" to false
  set visible of graphic "use tone" of card "main" to false
  set visible of graphic "gama on" of card "main" to false
  set visible of graphic "pre on" of card "main" to true
end mouseUp

on mouseEnter
  set visible of graphic "pre label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "pre label" to false
end mouseLeave
```

Object Script "BC" ID = 103
Friday, September 26, 1997 7:17 PM

```
on mouseUp
  set visible of graphic "gama on 2" to false
  set visible of graphic "pre on 2" to false
  set visible of graphic "levels on" to false
  set visible of graphic "curves on" to false
  set visible of graphic "use tone" of card "main" to false
  set visible of graphic "gama on" of card "main" to false
  set visible of graphic "pre on" of card "main" to false
end mouseUp

on mouseEnter
  set visible of graphic "BC label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "BC label" to false
end mouseLeave
```

Object Script "gama" ID = 107
Friday, September 26, 1997 7:17 PM

```
on mouseUp
  set visible of graphic "gama on 2" to true
  set visible of graphic "pre on 2" to false
  set visible of graphic "levels on" to false
  set visible of graphic "curves on" to false
  set visible of graphic "use tone" of card "main" to false
  set visible of graphic "gama on" of card "main" to true
  set visible of graphic "pre on" of card "main" to false
end mouseUp

on mouseEnter
  set visible of graphic "gama label" to true
end mouseEnter

on mouseLeave
  set visible of graphic "gama label" to false
end mouseLeave
```